โปรล็อก-ปาสคาล-โปรล็อก
การค้นในทางลึก

บุญเลิศ เอี่ยมทัศนา

ปิดม่าน การคิดเขียนโปรแกรมในภาษาโปรล็อกคิดเขียนได้ 2 ลักษณะคือ คิดเขียนในแบบแถลงความสัมพันธ์ (declarative) กับคิดเขียนในแบบกำหนดขั้นตอน (procedural) ซึ่งถึงแม้จะให้ผลออกมาเช่นเดียวกัน แต่ก็ยังเป็นที่ถกเถียงกันว่าแบบใดเหมาะสมที่สุด โดยทั่วไปเห็นว่าควรเขียนในแบบแถลงความสัมพันธ์ แต่ผู้ที่คิดเขียนในแบบนี้ส่วนมากก็คือผู้ที่เคยคิดเขียนโปรแกรมในแบบกำหนดขั้นตอนมาก่อน (เมื่อเขียนโปรแกรมในภาษาตามแบบ เช่น ปาสคาล ซี ฯลฯ) จึงมีสำนึกของการเขียนโปรแกรมในลักษณะกำหนดขั้นตอนอยู่ในตัว

ฉากสาม เขียนโปรแกรมด้วยภาษาโปรล็อกแล้ว ต้องการให้ทำงานเร็วขึ้นจะทำอย่างไร

ฉากสอง ทำไมเรียนภาษาโปรล็อกแล้วยังเขียนโปรแกรมไม่ได้

ฉากหนึ่ง ทำไมไม่นำเรื่องราวของภาษาโปรล็อกมากล่าวถึงบ้าง

เปิดม่าน

ผมเขียนบทความนี้ในแบบย้อนกลับ เมื่อมาถึงขั้นตอนนี้ก็ต้องเป็นเรื่องของการส่งกลับ จากคำถามที่ตั้งมาพอจะตอบได้ดังนี้

1. ภาษาโปรล็อกเป็นภาษาที่กว้างพอสมควร ใช้ได้ทั้งงานตามแบบและงานทางด้านปัญญาประดิษฐ์ งานตามแบบเช่นงานทางด้านฐานข้อมูลดังที่ใช้ดีเบส ซึ่งผมเห็นว่าผู้ที่เริ่มเรียนภาษาโปรล็อกก็พอมองเห็นว่า สามารถใช้ภาษาโปรล็อกได้โดยสะดวก จึงไม่ได้นำมาเขียน

ส่วนงานทางด้านปัญญาประดิษฐ์ ผู้เขียนจะต้องมีความรอบรู้หรือไม่ก็ต้องทำงานเป็นทีมกับผู้รู้ในวิทยาการสาขาใดสาขาหนึ่ง จึงไม่ได้นำมาเขียนอีกเช่นกัน หากจะเขียนก็คงจะต้องยกตัวอย่างเป็นเกมประเภททายปัญหาลับสมอง ซึ่งไม่ทราบว่าจะเป็นที่สนใจกันหรือไม่

2. ในการเขียนโปรแกรมภาษาโปรล็อก ผู้เขียนจะต้องเข้าใจกลไกภายในของโปรล็อก ซึ่งการทำความเข้าใจหากมีฐานความรู้เดิมมาจากภาษาตามแบบเช่นปาสคาล จะทำให้เข้าใจได้ง่าย ดังที่ผมจะนำมากล่าวในบทความชุดนี้จึงได้ชื่อว่า ปาสคาล –โปรล็อก

[**บทขัด** : กล่าวกันว่าผู้ที่ไม่เคยเรียนรู้ภาษาคอมพิวเตอร์ใดมาก่อน จะเข้าใจในภาษาโปรล็อกได้ดีกว่า แต่ก็มีแนวความคิดที่ขัดกันอยู่ และในปัจจุบันผู้ที่เขียนโปรแกรมภาษาโปรล็อกเก่ง ๆ ก็คือผู้ที่เรียนภาษาอื่นมาก่อนเกือบทั้งสิ้น]

3. ผมทราบมาว่าภาษาทางด้านปัญญาประดิษฐ์ เช่นโปรล็อกและลิสต์ เป็นภาษาของการทดสอบ เมื่อโครงงานสำเร็จแล้วมักจะนำไปเขียนด้วยภาษาตามแบบส่วนมากได้แก่ภาษาซี ทั้งนี้เพราะสามารถเขียนโปรแกรมในภาษาโปรล็อกได้สั้นกว่ากันมาก ทำให้คิดเขียนโปรแกรมได้เร็ว แต่ใช้เวลาในการรันมากกว่าโปรแกรมภาษาตามแบบ

แต่ในบทความชุดนี้ผมจะใช้ภาษาปาสคาล เพราะเห็นว่าภาษาปาสคาลเป็นภาษาของคำอธิบาย อ่านโปรแกรมได้ง่ายกว่าภาษาซี

โหมโรง

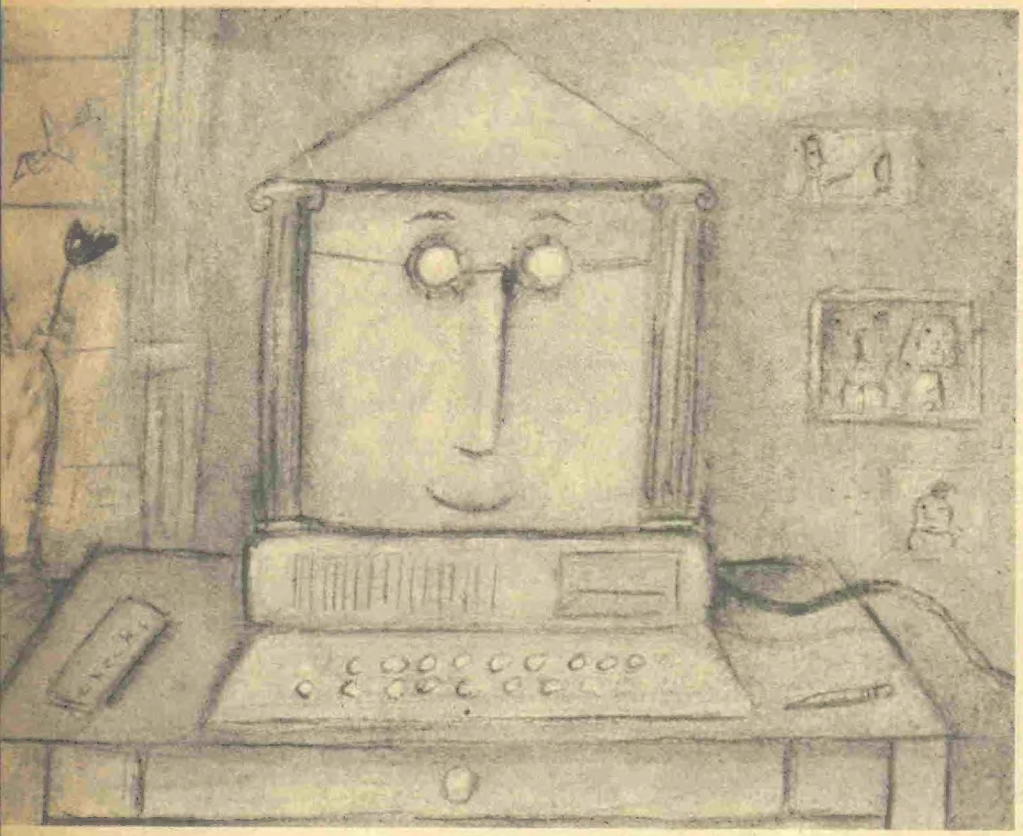
บทความชุดนี้จะเป็นการนำภาษาโปรล็อก(บางส่วน) มาศึกษาด้วยภาษาปาสคาล เพื่อนำความรู้ที่ได้ไปเขียนโปรแกรมทางด้านปัญญาประดิษฐ์ด้วยภาษาตามแบบ และนำความรู้นี้เองย้อนกลับไปใช้ในการเขียนโปรแกรมในภาษาโปรล็อกต่อไป

เริ่มที่โปรล็อก

หากคุณเขียนโปรแกรมภาษาโปรล็อกอย่างง่าย ๆ เช่นให้ข้อเท็จจริงไว้ว่า

```
road (a,b).
road (b,c).
road (c,d).
road (d,e).
```

หรืออาจจะต่อไปมากกว่านี้ โดยให้ความหมายว่ามีถนนจากเมือง a ไป b, b ไป c และต่อ ๆ ไป หากต้องการ



ทราบว่าจะมีเส้นทางจากเมืองหนึ่งไปยังอีกเมืองหนึ่งหรือไม่ เช่นนี้ก็สามารถคิดอย่างโปรล็อก (คิดอย่างแถลงความสัมพันธ์ และเป็นการคิดอย่างเป็นตรรก) ได้โดยตรง คือ

จะมีเส้นทางจากเมืองต้นทาง ไปเมืองปลายทาง ถ้ามีถนนจากเมืองต้นทางไปเมืองกลาง และมีถนนจากเมืองกลางไปเมืองปลายทาง

จะเห็นว่าเป็นความคิดอย่างเป็นตรรก โดยไม่ต้องอาศัยความรู้เดิมในภาษาตามแบบมาก่อน แต่เมื่อนำไปเขียนโปรแกรม จะต้องเขียนในแบบเรียกตนเอง (recursion) คือ

```
route (D,D) :-!
route (S,D) :- road (S,M),
               route (M,D).
```

ถึงตรงนี้ครับถกเถียงกันได้แล้วว่า การเขียนโปรแกรมในแบบเรียกตนเองนี้ใช้ความรู้เดิมหรือไม่ ซึ่งพอจะออมชอมกันได้ว่า "การเรียกตนเอง" เป็นของกลาง จึงอาจเป็นได้ทั้งใช้ความรู้เดิมหรือไม่ใช้

เมื่อรันโปรแกรมนี้ เครื่องจะตอบว่า yes หรือ no เท่านั้น หากคุณไม่สงสัยว่าเครื่องทำงานได้อย่างไรก็ไม่ต้องสนใจกลไกภายในของโปรล็อก แต่ถ้าคุณต้องการให้เครื่องแสดงชื่อเมืองในเส้นทางด้วย ตรงนี้แหละครับที่ต้องทราบกลไกภายในของโปรล็อกเพราะต้องนำมาใช้ ดังที่ผมเคยกล่าวว่า การเขียนโปรแกรมในภาษาโปรล็อกเหมือนกับการเขียนโปรแกรมบนโปรแกรม

กลไกภายในของโปรล็อก

กลไกภายในของโปรล็อกที่สำคัญมีอยู่ 3 เรื่องคือ

● ฐานข้อมูล

ฐานข้อมูลเป็นที่เก็บข้อมูล ซึ่งในภาษาโปรล็อกได้แก่ ข้อเท็จจริง (facts) และกติกา (rules) ตามแต่ผู้ใช้จะป้อนและตามที่โปรแกรมจะสร้างขึ้นใหม่ เรียกว่าเป็นฐานข้อมูล (Data bases) เมื่อมองในแง่ของงานตามแบบหรือเรียกว่าฐานความรู้ (Knowledge bases) เมื่อมองในแง่ของงานทางด้านปัญญาประดิษฐ์

ในโปรล็อกจะต้องมีวิธีดำเนินการกับฐานข้อมูลอย่างมีประสิทธิภาพ อันขึ้นอยู่กับรูปแบบของการเก็บข้อมูลด้วย แต่ผมไม่ทราบว่าในโปรล็อกเก็บข้อมูลอย่างไร ทราบมาอย่างไม่แน่ใจนักว่าเก็บในรูปของอะเรย์ของอักขระ ซึ่งต้องเขียนโปรแกรมยาวมากในการดำเนินกรรมวิธีกับข้อมูล ในบทความนี้จึงจะเก็บในรูปอะเรย์ของเรคอร์ด ทำให้เขียนโปรแกรมได้สั้นลง แต่ใช้หน่วยความจำเปลืองกว่า

● การค้นหาคำตอบ (Search for solution)

การค้นหาคำตอบ แตกต่างจากการค้นหาข้อมูล (search) ดังเช่นตัวอย่างเรื่องเส้นทางที่กล่าวมาเป็นการค้นหาคำตอบ ส่วนการค้นหาข้อมูลเป็นเพียงแต่ค้นหาข้อมูลใดข้อมูลหนึ่ง ในฐานข้อมูลโดยไม่คิดถึงความสัมพันธ์กัน หรือกล่าวอีกนัยหนึ่งการค้นหาคำตอบคือการค้นหาข้อมูลที่มีความสัมพันธ์กัน

วิธีการค้นหาคำตอบของโปรล็อกเป็นเรื่องที่ผู้เขียนโปรแกรมในภาษานี้จำเป็นต้องทราบ เพื่อนำไปใช้ในการเขียนโปรแกรม ซึ่งการค้นหาคำตอบในตัวของโปรล็อกเอง ใช้วิธีค้นหาในทางลึก แต่ผมจะนำการค้นหาคำตอบที่นิยมใช้กันมากมากล่าวถึงด้วย ทั้งนี้เพื่อจะนำไปใช้ในการเขียนโปรแกรมทั้งในภาษาโปรล็อกและภาษาตามแบบ คือ

1. การค้นหาอย่างเป็นระบบ (Systemmatic Searches)

1.1 ค้นในทางลึก (Depth-First Search)

1.2 ค้นในทางกว้าง (Breadth-First Search)

2. Heuristic Searches
2.1 Hill Climbing Search
2.2 Least-Cost Search

● การดำเนินกรรมวิธีกับลิสต์ (List Processing)

ผมทราบมาอย่างไม่ค่อยแน่ใจนักอีกเช่นกันว่า การเก็บข้อมูลในแบบลิสต์ของโปรล็อก เก็บอยู่ในรูปของสแต็กแต่มองกลับกัน ซึ่งเรื่องนี้ต้องการความรู้พื้นฐานในเรื่องลิสต์ คิว และสแต็ก ของภาษาตามแบบก่อน ดังที่จะขอนำมากล่าวโดยย่อต่อไปนี้

ลิสต์ คิว และสแต็ก

ในภาษาตามแบบข้อมูลที่เก็บอยู่ในลิสต์ ในคิว หรือในสแต็ก ความจริงแล้วคือข้อมูลที่เก็บอยู่ในอะเรย์ทั้งสิ้น แตกต่างกันตรงการมองและการใช้งาน กล่าวคือ

ลิสต์ (list)

การเก็บข้อมูลในแบบลิสต์ จะเก็บเรียงกันไปดังแสดงในรูปที่ 1 (รูปที่ 1 ถึงรูปที่ 4 สมมติว่าได้ป้อนข้อมูลเรียงลำดับจาก a ถึง f) มีอินเด็กซ์หนึ่งตัว (last) ชี้ข้อมูลที่อยู่ท้ายลิสต์ มีอินเด็กซ์อีกหนึ่งตัว (idx) หรือหลายตัวไว้ชี้หรือค้นหาข้อมูล

การเพิ่มข้อมูลโดยปกติจะนำมาต่อท้ายลิสต์ การถอนข้อมูลวิธีที่ง่ายคือนำข้อมูลตัวสุดท้ายมาไว้ตรงตัวที่จะถอนแล้วลดอินเด็กซ์ท้ายลิสต์ลงหนึ่ง เช่นตามรูปที่ 1 หากต้องการถอน b จะเขียนส่วนของโปรแกรมได้ดังนี้

```
list [idx] := list [last];
last := last−1;
```

คิว (Queue)

เปรียบเทียบได้เช่นเดียวกับการเข้าคิวดังแสดงในรูปที่ 2 มีอินเด็กซ์อย่างน้อย 2 ตัว ตัวหนึ่งไว้ชี้ต้นคิว อีกตัวหนึ่งไว้ชี้ท้ายคิว ตามรูปคือ h และ t ตามลำดับ เริ่มต้นด้วยการให้อินเด็กซ์ทั้งคู่นี้ชี้ที่ช่องเดียวกัน

เมื่อต้องการเพิ่มข้อมูลให้แก่คิวจะให้เก็บตรงช่องที่ t ชี้อยู่แล้วเพิ่มค่า t ขึ้นหนึ่ง นั่นคือจะเพิ่มข้อมูลต่อท้ายคิวเสมอ เมื่อต้องการถอนข้อมูลจะอ่านข้อมูลตรงช่องที่ h ชี้อยู่แล้วเพิ่มค่า h ขึ้นหนึ่ง นั่นคือจะถอนข้อมูลจากหัวคิวเสมอ (ตามรูปแสดงว่าได้ถอนข้อมูล a จากคิวไป 1 ข้อมูล) หาก h และ t เท่ากันอีกแสดงว่าคิวว่างเปล่า

สแต็ก (Stack)

เทียบได้กับการวางของเช่นจานซ้อนทับกัน ดังนั้นจานที่จะหยิบได้เป็นใบแรกคือจานที่วางเป็นใบสุดท้าย มีลักษณะการเก็บดังแสดงในรูปที่ 3 โดยมีอินเด็กซ์ตัวหนึ่งตามรูปคือ SP ชี้ที่ท้ายอะเรย์แต่เรียกเป็นชั้นบนของสแต็ก

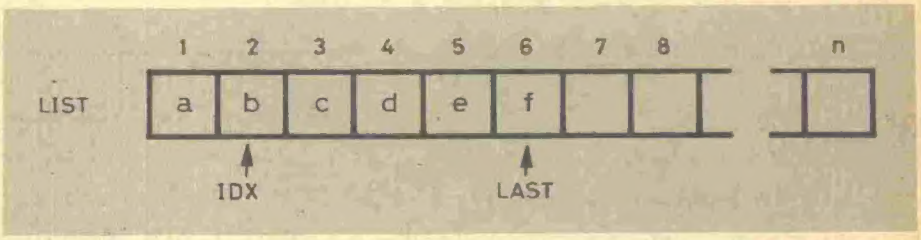
โดยทั่วไปจะนิยมเริ่มด้วยให้ค่า SP น้อยกว่า(หรือมากกว่า)ช่อง เช่นตามรูปจะให้ SP = 0 เมื่อต้องการเก็บข้อมูลลงสแต็กคือพุช (push) จะเพิ่มหนึ่ง SP แล้วเก็บข้อมูลลงช่องที่ SP ชี้อยู่ เมื่อต้องการถอนข้อมูลจากสแต็กคือป๊อป (pop) จะอ่านข้อมูลตรงช่องที่ SP ชี้อยู่แล้วลดค่า SP ลงหนึ่ง ดังนั้นการเพิ่มหรือถอนข้อมูลจากสแต็กจะทำที่ชั้นบนสุดของสแต็กเท่านั้น

ลิสต์ในโปรล็อก

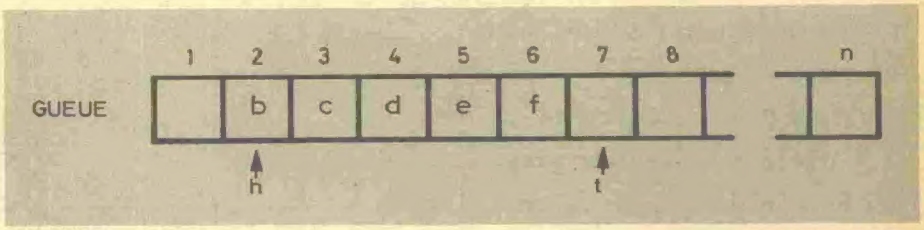
ลิสต์ในโปรล็อกมีลักษณะเช่นเดียวกับสแต็กในภาษาตามแบบ แต่ต้องมองกลับกับสแต็กดังแสดงในรูปที่ 4 จะเป็นลิสต์ของ

[f,e,d,c,b,a]

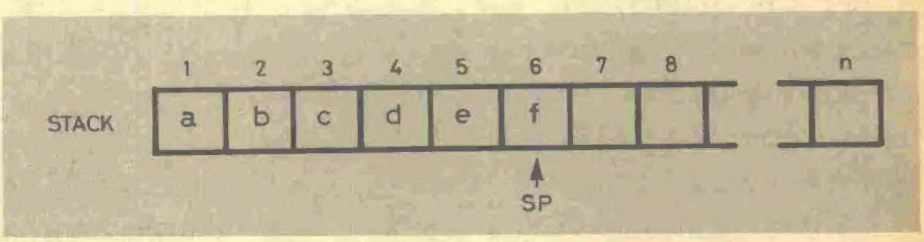
โดยที่มี f เป็นส่วนหัว (head) และข้อมูลที่เหลือเป็นส่วนหาง (tail)



รูปที่ 1 ข้อมูลที่เก็บในรูปของลิสต์



รูปที่ 2 ข้อมูลที่เก็บในรูปของคิว



รูปที่ 3 ข้อมูลที่เก็บในรูปของสแต็ก

การเพิ่มและถอนข้อมูลกับลิสต์จะดำเนินการตรงส่วนหัวเท่านั้น เทียบ head ได้กับ SP แต่กลับข้างกันเพื่อต้องการให้เห็นเหมือนกับที่เขียนค่าในลิสต์ แต่ความเป็นจริงแล้วก็คือสแต็กนั่นเอง

การค้นหาในทางลึก (Depth-First Search)

สมมติว่าเรามีแผนที่ทางหลวงระหว่างเมืองดังแสดงในรูปที่ 5 แต่เป็นทางไปทางเดียว (one way) ดังที่ทำเป็นหัวลูกศรไว้ (ดูจะไม่สมเหตุผลนักแต่ขอให้คิดว่าเป็นตัวอย่างขั้นต้นครับ) ให้เขียนโปรแกรมในภาษาปาสคาล ให้ผู้ใช้ถามว่ามีเส้นทางจากเมืองหนึ่งไปยังอีกเมืองหนึ่งหรือไม่ โดยให้เครื่องตอบว่า yes หรือ no (ดังที่เขียนเป็นโปรแกรมภาษาโปรลือกไว้แล้ว)

จะเห็นว่าโจทย์เช่นนี้จะต้องดำเนินการด้วยการค้นหาคำตอบ ซึ่งผมจะเลือกเอาวิธีค้นหาในทางลึกมากล่าวถึงก่อน

กรรมวิธี

การค้นหาในแบบนี้จะเริ่มจากเมืองต้นทาง ถ้ามีทางแยกจะแยกไปทางซ้ายก่อน ค้นไปจนสุดทางหากไม่พบจะย้อนกลับมาตรงทางแยก ค้นไปทางขวาจนสุดทาง หากไม่พบจะทำซ้ำจนพบหรือสุดฐานข้อมูล

เช่นหากถามว่ามีเส้นทางจาก a ไป h หรือไม่ การค้นหาในทางลึกจะดำเนินการดังนี้คือ ค้นไปในเส้นทาง a-b-c-d-e ย้อนกลับมา d ค้น f ย้อนกลับมา d ค้น i-j-k ไม่พบ ย้อนกลับมา a (คือทางแยก) ค้นในเส้นทาง a-x-y และ a-x-z ไม่พบค้นในเส้นทาง a-h พบ ตอบว่า yes (จะเห็นว่าได้ค้นไปในเส้นทางที่วกวนมาก และนี่คือข้อเสียของการค้นในทางลึก)

ที่กล่าวว่าจะค้นไปทางซ้ายก่อนเป็นเรื่องของการเขียนภาพกับการป้อนข้อมูล หากต้องการให้เป็นไปตามที่กล่าวมานี้ จะต้องป้อนเส้นทาง a-b ไว้ก่อน a-x ไว้ก่อน a-h หากป้อนเส้นทาง a-h ก่อน ก็จะต้องเขียนเส้นทาง a-h ไว้ทางซ้าย หากใช้รูปเดิมก็จะเป็นการค้นไปทางขวาก่อน

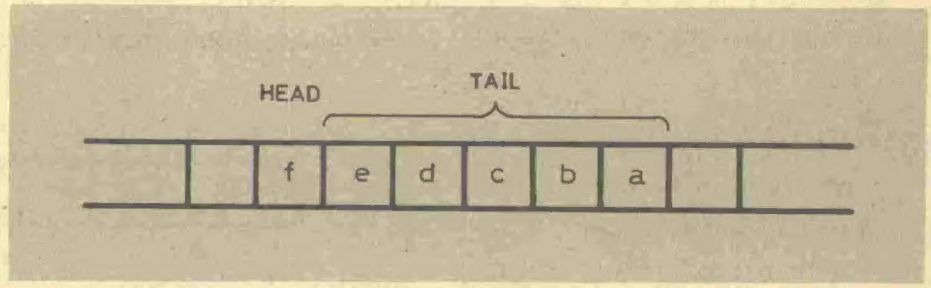
การย้อนรอย

ตามตัวอย่างที่ยกมานี้ได้กล่าวว่าการค้นในทางลึกจะค้นในเส้นทาง a ถึง e ก่อน แล้วย้อนกลับมาที่ d (แต่ต่อมาได้ย้อนกลับมาที่ a) การย้อนกลับนี้ในโปรลือกจะต้องมีกลไกภายในที่ต้องดำเนินการอยู่ 3 เรื่อง คือ

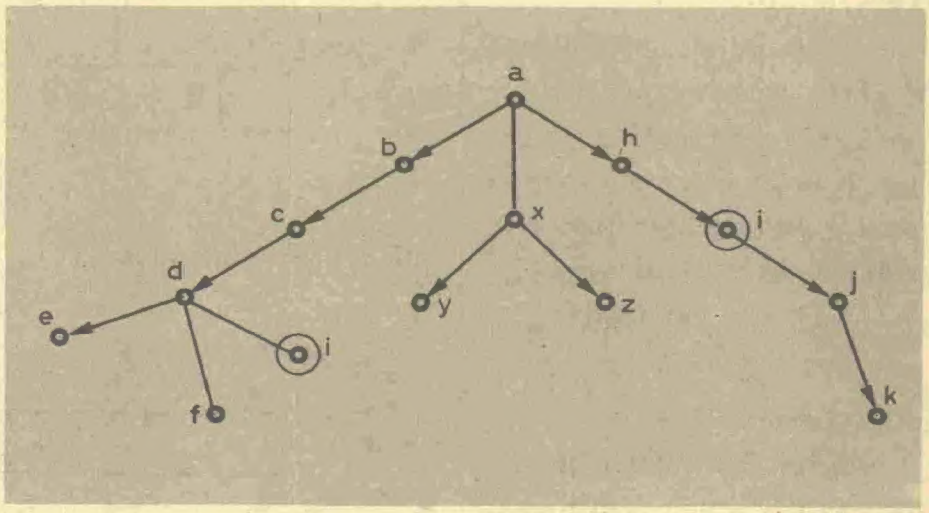
- จะต้องย้อนกลับในเส้นทางเดิมเท่านั้น
- จะต้องหาทางแยก เพื่อเดินไปตามทางแยกใหม่ด้วย
- จะต้องไม่ย้อนกลับไปในทางแยกเก่า เช่น ที่จุด d เมื่อไป e แล้วจะต้องหาทางไป f และ i โดยไม่ย้อนกลับไป e อีก

การดำเนินการทั้ง 3 เรื่องนี้รวมเรียกว่า back tracking ซึ่งเมื่อผมเขียนหนังสือ "เรียนรู้ภาษาโปรลือก" ได้พยายามหาคำไทยที่เหมาะสม รวมทั้งคำว่า "ย้อนรอย" ด้วย แต่ไม่ได้นำมาใช้เพราะชวนให้คิดว่าเป็นการย้อนกลับแต่เพียงอย่างเดียว จึงใช้คำกลาง ๆ ว่า "การกระทำซ้ำ" แต่ในตอนนี้มีผู้ใช้มากขึ้นก็ควรจะใช้ให้เหมือนกันครับ

การย้อนรอยดังกล่าวมานี้ใน 2



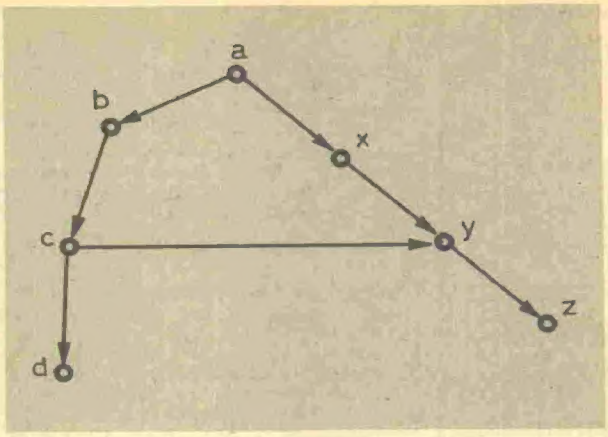
รูปที่ 4 ข้อมูลที่เก็บในรูปของลิสต์ในภาษาโปรลือก โดยสมมติว่าให้ค่าตั้งแต่ a ถึง f แก่ลิสต์ตามลำดับ



รูปที่ 5 แสดงแผนที่ทางหลวงระหว่างเมืองสมมติ ขอให้สังเกตว่า มีเส้นทางต่อจาก d ไป i แต่ไม่ได้โยงเส้นตรงเพราะต้องการเขียนในแบบ directed graph

เรื่องแรกพอมองเห็นว่าในโปรล็อกใช้การเรียกตนเอง (ดังจะเห็นในโปรแกรมภาษาปาสคาล แต่การป้องกันไม่ให้ย้อนกลับไปทางแยกเก่า ผมไม่ทราบว่าในโปรล็อกดำเนินการอย่างไร สำหรับในโปรแกรมภาษาปาสคาลผมจะใช้วิธีที่นิยมกันพอสมควรคือ การหมายรอยเท้าไว้ (วิธีนี้มีข้อเสียคือ ทุกครั้งที่ตั้งคำถามใหม่ จะต้องลบรอยเท้าเดิมก่อนเสมอ หากฐานข้อมูลใหญ่จะต้องใช้เวลานาน)

หมายเหตุ บางท่านอาจติดใจสงสัยว่าการค้นหาในทางกว้างมีลักษณะเป็นอย่างไร ผมจึงขอนำมากล่าวโดยสังเขป คือ ขอให้ดูในรูปที่ 5 หากเป็นการค้นหาในทางกว้าง จะตรวจว่าเมืองปลายทางเป็นเมือง b,x และ h หรือไม่ ถ้าไม่ใช่ก็จะตรวจว่าเป็นเมืองc,y,z และ i หรือไม่ คือลึกลงไปอีกระดับหนึ่ง ถ้าไม่ใช่ก็ตรวจในระดับต่อ ๆ ไป



รูปที่ 6 แสดงแผนที่เส้นทางสมมติ

```
/* Depth-First Search */
domains
  city = symbol
predicates
  road(city,city)
  route(city,city)
clauses
  road(a,b).
  road(b,c).
  road(c,d).
  road(a,x).
  road(x,y).
  road(y,z).
  road(c,y).

  route(D,D).
  route(S,D) :-  road(S,M),route(M,D).
```

รูปที่ 7 โปรแกรมในภาษาโปรล็อก ที่ให้คำตอบว่าเป็นจริงหรือเท็จเท่านั้น

โปรแกรมในภาษาปาสคาล

ผมจะเขียนโปรแกรมในภาษาปาสคาลเพื่อแสดงว่าโปรล็อก(โดยเฉพาะเทอร์โบโปรล็อก) ดำเนินการอย่างไรในการค้นหาคำตอบ (คือจะเป็นการค้นหาในทางลึกก่อนจริงหรือไม่) แต่เพื่อความง่ายในการเปรียบเทียบกัน จะขอเปลี่ยนแผนที่ทางหลวงเป็นดังรูปที่ 6

และได้เขียนโปรแกรมในภาษาโปรล็อกเพื่อใช้กับเทอร์โบโปรล็อกไว้ในรูปที่ 7 โปรแกรมนี้จะบอกว่ามีเส้นทางจากเมืองหนึ่งไปยังอีกเมืองหนึ่งหรือไม่เท่านั้น โดยแบ่งออกเป็น 2 ส่วน (ได้หัวข้อ Clauses) ส่วนแรกเป็นการให้ค่าแก่ฐานข้อมูล ส่วนที่สองให้โปรล็อกค้นหาคำตอบ

สำหรับในภาษาปาสคาลข้อมูลที่จะให้ในโปรแกรม เนื่องจากเราดำเนินการกับถนนเพียงอย่างเดียว จึงจะตัดคำว่า road ออกคือแทนที่จะให้เป็น road (a,b) จะให้เป็น a,b เท่านั้น และเพื่อเตรียมการไว้จะให้มีระยะทางด้วย คือจะให้ข้อมูลเป็น a,b,20. หมายถึงถนนจากเมือง a ไป b ระยะทาง 20 กิโลเมตร เครื่องหมายจุดเพื่อใช้ในโปรแกรม

แบบของข้อมูลและตัวแปร

ดังการกำหนดแบบข้อมูลและตัวแปรในรูป A แบบข้อมูล RoadT ประกอบด้วย ฟิลด์ frm หมายถึงเมืองต้นทาง ฟิลด์ too หมายถึงเมืองปลายทาง

A

```
PROGRAM  DepthFirstSearch(input,output);
USES     CRT;
CONST    WRDLEN = 20;  MAX = 100;  ESC = #27;
TYPE     NameT = String[WRDLEN];
         RoadT = RECORD
           frm  : NameT;
           too  : NameT;
           dist : Word;
           pass : Boolean;
                  END;
VAR      roads : ARRAY[1..MAX] OF RoadT;
         stack : ARRAY[1..MAX] OF NameT;
         RP, SP : Word;  quit, found : Boolean;
         rdestr : String;  lstch : Char;

{*************************************************************}

PROCEDURE  GetSentence(var astr : String; var ch : Char);
VAR      len : Byte;
BEGIN
  ReadLn(astr);
  len := Length(astr);  ch := astr[len];
  IF  ch IN ['!','?']  THEN  astr[len] := '.'
END;
```

ฟิลด์ dist หมายถึงระยะทาง และฟิลด์ pass หมายถึงได้เดินผ่านเมืองนี้หรือไม่ เพื่อการหมายรอยเท้า

นำไปกำหนดตัวแปร roads เป็นอะเรย์ของถนนคือ RoadT ขนาด 100 ช่อง ตัวแปร roads นี้คือฐานข้อมูลหรือฐานความรู้สุดแต่จะมอง (ซึ่งจะเก็บข้อมูลถนนได้ไม่เกิน 100 สาย) ตัวแปร stack และ SP ยังไม่ใช้ในตอนนี้ สำหรับ RP จะใช้เป็นอินเด็กซ์ของตัวแปร roads

ในโปรแกรมภาษาโปรล็อกให้ข้อมูลตรง ๆ เช่นที่เขียนว่า road (a,b). จะมีโปรแกรมย่อยตัวหนึ่งดำเนินการในเรื่องนี้ ผมจึงเขียนโพรซิเดอร์ Assert Data ดังแสดงในรูป B

AssertData

โพรซิเดอร์นี้จะแปลงสตริงเช่น 'a,b,20.' เป็นข้อมูลของถนน ให้แก่ฐานข้อมูลตรงช่อง RP+1 (ค่า RP จะให้โปรแกรมหลักให้ค่าเป็น 0 แต่เริ่มต้น) ด้วยการเรียกใช้ GetWord โดยที่ GetWord จะอ่านคำจากสตริงมาให้คำหนึ่ง โดยถือว่า คำจะขึ้นต้นด้วยอักขระที่ไม่ใช่อักขระวรรค ไปจนถึงพบจุดหรือวรรคอีกครั้งหนึ่ง (ไม่รวมวรรคและจุดเข้ากับคำ)

DoAssert

แต่โปรแกรมที่ทำหน้าที่ให้ข้อมูลกับฐานข้อมูลแท้ ๆ ตรงกับที่เขียนในภาษาโปรล็อกคือ DoAssert อยู่ในรูป C

SeachRoad

ขอให้ดูโปรแกรมในภาษาโปรล็อกอีกครั้ง หลังจากส่วนที่ให้ข้อมูลกับฐานข้อมูลแล้ว คือส่วนที่ให้โปรล็อกค้นหาคำตอบ ส่วนหลังนี้ผมนำมาเขียนเป็นโปรแกรมในภาษาปาสคาล คือโพรซิเดอร์ SearchRoad รูป E1

B

```
FUNCTION  GetWord(astr : String; var i : Byte) : NameT;
VAR  n : Byte;
BEGIN
  n := 1;
  while  astr[i] = ' '  do  i := i + 1;
  WHILE  (astr[i] <> ' ') AND (astr[i] <> '.')  DO
    BEGIN  GetWord[n] := astr[i];  Inc(n);  Inc(i)  END;
  GetWord[0] := Chr(n - 1);
END;

PROCEDURE  AssertData(astr : String);
VAR    idx : Byte;  dist : NameT; res : Integer;
BEGIN
  idx := 1;  RP := RP + 1;
  roads[RP].frm := GetWord(astr,idx);
  roads[RP].too := GetWord(astr,idx);
  dist := GetWord(astr,idx);
  Val(dist,roads[RP].dist,res);
  IF  res <> 0  THEN
    BEGIN  RP := RP - 1;
           WriteLn('Error in Integer number')  END;
END;
```

C

```
PROCEDURE  DoAssert;
BEGIN
  AssertData('a b 1.');
  AssertData('b c 2.');
  AssertData('c d 3.');
  AssertData('a x 4.');
  AssertData('x y 5.');
  AssertData('y z 6.');
  AssertData('c y 7.');
END;
```

E1

```
FUNCTION  FindMid(src : NameT; var mid : NameT;
                                var idx : Word) : Boolean;
VAR   i : Word;
BEGIN
  FindMid := TRUE;
  FOR  i := 1 TO RP  DO
    IF  (roads[i].frm = src) AND (roads[i].pass = FALSE)  THEN
      BEGIN  mid := roads[i].too;  idx := i;
             roads[i].pass := TRUE;  Exit                    END;
  FindMid := FALSE;
END;


PROCEDURE  SearchRoad(src,dst : NameT);
VAR  dist : Word;  mid : NameT;  idx : Word;
BEGIN
  found := FALSE;
  IF  src = dst  THEN  found := TRUE
  ELSE  IF  FindMid(src,mid,idx)  THEN
    SearchRoad(mid,dst);

  IF  NOT found  THEN
    IF  MoreRoad(src)  THEN  SearchRoad(src,dst);
END;


PROCEDURE  DoQuestion(astr : String);
VAR   srce,dstn : NameT;  dist : Word; idx : Byte;
      aroad : Boolean;
BEGIN
  ClearPass;  idx := 1;  aroad := FALSE;
  srce := GetWord(astr,idx);
  dstn := GetWord(astr,idx);
  SearchRoad(srce,dstn);
  IF  found  THEN  WriteLn('yes')
  ELSE  WriteLn('no');
END;
```

โปรแกรมนี้จะค้นหาว่ามีเส้นทางจากเมืองต้นทางคือ src ไปยังเมืองปลายทางคือ dst หรือไม่ ถ้ามีจะให้ found เป็นจริง ถ้าไม่มีให้เป็นเท็จ โดยเขียนตรงตามโปรแกรมภาษาโปรล็อกคือ

- ถ้าเมืองต้นทาง เป็นเมืองเดียวกับเมืองปลายทาง ถือว่ามาถึงเมืองปลายทางแล้ว ให้ found เป็นจริงคือที่เขียนในภาษาโปรล็อกว่า

route (D,D).

- ถ้าไม่ใช่ให้หาว่ามีเมืองกลางหรือไม่ คือ FindMid จะตรงกับภาษาโปรล็อกว่า

road (S,M).

- ถ้ามีให้เรียกตนเอง โดยนำเมืองกลางมาเป็นเมืองต้นทาง ตรงกับภาษาโปรล็อกว่า

route (M,D).

- แต่ถ้าไม่มีเมืองกลางก็จะเป็นการส่งกลับ(return) จากการเรียกตนเองจะนำมาตรวจดูว่าถึงเมืองปลายทางหรือไม่ ถ้ายังไม่ถึงจะตรวจหาทางแยกด้วย MoreRoad ถ้ามีทางแยกก็จะเรียกตนเองอีก

ขอให้ดูในรูปที่ 6 สมมติว่าถามเส้นทางจากเมือง a ไป x การดำเนินการมาถึงจุดนี้คือเดินมาถึงเมือง d ปรากฏว่าไม่มีเมืองกลางจึงส่งกลับไปเมือง c หาทางแยกไป y ได้จึงเรียกตนเองไป y, z ปรากฏว่าตันอีกจึงส่งกลับไป y, c, b ไม่มีทางแยกเลยจนถึง a ปรากฏว่ามีทางแยกไป x และเป็นเมืองปลายทาง ทำให้ found เป็นจริง ต่อไปเป็นการส่งกลับซึ่งไม่มีการดำเนินการใด จบจากโพรซิเดอร์นี้

FindMid (รูป E1)

ในโพรซิเดอร์ SearchRoad ได้เรียกใช้ฟังก์ชัน FindMid เพื่อให้ค้นหาเมืองกลางให้ ซึ่ง FindMid จะให้ค่าความเป็นจริงเมื่อค้นพบ และให้ชื่อเมืองกลางในตัวแปร Mid หากค้นไม่พบจะให้ค่าความเป็นเท็จ

การค้นหาข้อมูลของ FindMid จะค้นจากต้นจนพบหรือจนสุดฐานข้อมูลทุกครั้ง ทั้งนี้เพราะการให้ค่าข้อมูลนั้นจะให้ค่าสลับกันอย่างไรก็ได้ (และในเทอร์โบโปรล็อกก็ดำเนินการเช่นนี้ ตรวจดูได้ด้วยการแทรกชิง)

ในการค้นหาจะค้นหาเมืองต้นทางซึ่งส่วนใหญ่แล้วคือเมืองกลางจากการเรียกตนเอง เมื่อพบจะตรวจดูว่าเป็นเมืองที่เคยผ่านมาแล้วหรือไม่ (คือได้หมายรอยเท้าไว้แล้วหรือไม่) ถ้ายังไม่เคยผ่านจะให้ชื่อใน too แก่ mid แล้วหมายรอยเท้าไว้คือให้ pass เป็นจริง

MoreRoad (รูป D)

ดำเนินการเช่นเดียวกับ FindMid แต่เพียงตรวจดูว่ามีทางแยก (ซึ่งความจริงก็คือเมืองกลางนั่นเอง) หรือไม่ และไม่ได้หมายรอยเท้า

DoQuestion (รูป E1)

ในการทำงานกับโปรล็อกนั้น ผู้ใช้จะต้องตั้งคำถามให้เครื่องตอบ ซึ่งในโปรล็อกจะมีโปรแกรมตัวหนึ่งดำเนินการในเรื่องนี้ จึงได้เขียนโปรแกรม DoQuestion เลียนแบบ

โปรแกรมนี้จะนำสตริงคือคำถามที่โปรแกรมหลักส่งมา เช่น ad .มาแยกเป็นเมืองต้นทางและปลายทาง แล้วเรียกใช้ SearchRoad หากได้ found เป็นจริงก็จะตอบว่า yes ไม่เช่นนั้นจะตอบว่า no

ตอนต้นของโปรแกรมได้เรียกใช้ ClearPass (อยู่ในรูป D) เพื่อลบรอยเท้าอันเกิดจากคำถามครั้งก่อน ดังได้กล่าวแล้วว่าถ้าฐานข้อมูลใหญ่จะทำให้เสียเวลา

โปรแกรมหลัก

ได้เขียนโปรแกรมหลักไว้ในรูป M โดยกำหนดว่าให้ผู้ใช้ตั้งคำถามโดยระบุชื่อเมืองต้นทางวรรคเมืองปลายทางปิดท้ายด้วยเครื่องหมาย ? (เพราะต่อไปจะปรับปรุงโปรแกรมนี้ให้สามารถป้อนข้อมูลได้โดยใช้เครื่องหมาย . ปิดท้าย ใช้คำสั่งได้โดยใช้ ! ปิดท้าย) แต่ GetSentence (อยู่ในรูป A) จะเปลี่ยนเป็น . ทั้งหมด เพื่อใช้ในโปรแกรม

การทำงานของโปรแกรมหลักเริ่มด้วยการลบจอภาพให้ค่า RP เป็น 0 ให้ข้อมูลแก่ฐานข้อมูล แล้วเข้าทำงานในวงรอบ REPEAT แสดงเครื่องหมาย พร้อมเป็น : เพื่อให้ผู้ใช้ตั้งคำถาม อ่านให้แก่ตัวแปร rdestr ส่วน lstch จะเป็นอักขระตัวสุดท้ายของสตริงนำมาตรวจถ้าเป็น ? ก็ให้ตอบคำถาม ทำซ้ำจนกว่าผู้ใช้จะป้อนคำว่า quit. หรือ quit! ก็ได้

D

```
PROCEDURE  ClearPass;
VAR  i : Word;
BEGIN
  FOR  i := 1 TO RP  DO  roads[i].pass := FALSE;
END;

FUNCTION  MoreRoad(src : NameT) : Boolean;
VAR  i : Word;
BEGIN
  MoreRoad := TRUE;
  FOR  i := 1 TO RP  DO
    IF  (roads[i].frm = src) AND (roads[i].pass = FALSE)  THEN
      Exit;
  MoreRoad := FALSE;
END;
```

กลับไปที่โปรล็อก

จากที่กล่าวมานี้จะเห็นว่า

- โปรล็อกค้นหาคำตอบด้วยวิธีค้นในทางลึกเป็นหลัก (เพราะจะมีสิ่งอื่นมาแทรกดังจะชี้ให้เห็นต่อไป)

- การค้นหาข้อมูลในการค้นในทางลึก โปรล็อกจะค้นจากต้นฐานข้อมูลจนพบหรือจนสุดฐานข้อมูลทุกครั้ง ดังแสดงในโพรซิเดอร์ FindMid (ที่ว่าค้นจากต้นฐานข้อมูลทุกครั้ง มีข้อยกเว้นดังจะชี้ให้เห็นต่อไป)

- ในการค้นหาข้อมูลนี้ มีการย้อนรอยคือ ค้นไปทางซ้ายจนสุดทาง หากไม่พบจะย้อนกลับมาทีละเมือง พร้อมกับหาทางแยกใหม่ไปทางขวา หากพบจะดำเนินการเช่นเดียวกัน คือ ค้นไปจนสุดทางแยกนั้น

เพื่อความเข้าใจขอให้แทรกคำสั่ง Write (src) นี้ ไว้ที่ต้นโพรซิเดอร์ SearchRoad แล้วทดลองรันดู เช่นป้อนคำถามเป็น a x? จะเห็นวิธีค้นหาในทางลึกว่าดำเนินการอย่างไร แล้วทดลองรันโปรแกรมภาษาโปรล็อก (รูปที่ 7) ด้วยการแทรกซิง (เรียนรู้ภาษาโปรล็อก หน้า 155) เปรียบเทียบกัน

จากโปรแกรมทั้ง 2 ภาษานี้จะเห็นว่า เราเขียนโปรแกรมในภาษาโปรล็อกได้ง่ายและสั้นกว่าในภาษาปาสคาลมาก คือเราไม่ต้องนึกถึงโครงสร้างข้อมูลของฐานข้อมูล ไม่ต้องนึกว่าจะค้นหาคำตอบได้อย่างไร ปล่อยให้เป็นภาระของโปรล็อกดำเนินการ

สแต็กในปาสคาล

โปรแกรมภาษาโปรล็อกที่กล่าวมานี้ยังใช้งานได้น้อยมาก เพราะเพียงแต่ตอบว่า TRUE หรือ FALSE เท่านั้น หากต้องการให้แสดงชื่อเมืองที่ผ่านด้วย เราจะต้องดำเนินการเอง แต่ขอปรับปรุงโปรแกรมในภาษาปาสคาลก่อนคือ

ให้ใช้โพรซิเดอร์ SearchRoad ในรูป E2 แทนของเดิม ในโพรซิเดอร์นี้ได้นำสแต็กสมมติคือตัวแปร stack มาใช้ โดยให้ตัวแปร sp (อ่านว่า stack pointer) เป็นตัวชี้ข้อมูลใน stack และจะให้ DoQuestion ให้ค่า SP เป็น 0 แต่เริ่มต้น

การทำงานของโพรซิเดอร์นี้ในส่วนที่เพิ่มขึ้นมาคือ เช่นเราถามเส้นทางเป็น a d? ก็จะมีการเก็บชื่อเมือง a, b, c และ d ไว้ในสแต็กของซีพียูตามลำดับ เมื่อถึงเมือง d จะได้ found เป็นจริง ดังนั้นในขั้นตอนของการส่งกลับ ที่ให้ตรวจว่า IF found THEN ก็จะเป็นจริง นำค่าที่ส่งกลับมาซึ่งจะเรียงลำดับกลับเป็น d, b, c และ a มาเก็บในสแต็กสมมติ คือตัวแปร stack จึงได้ว่าเมือง a อยู่ที่ชั้นยอดของสแต็กสมมติ ซึ่ง DoQuestion จะรับไปดำเนินการต่อ

จึงต้องเปลี่ยนโปรแกรม DoQuestion เป็นตามรูป F ที่เพิ่มขึ้นมา คือเงื่อนไขที่ให้ตรวจว่าชื่อเมืองต้นทางกับเมืองปลายทางเป็นชื่อเดียวกันหรือไม่ เช่นผู้ใช้อาจถามว่า bb ? สำหรับวงรอบ while MoreRoad (srce) Do จะให้ sp เป็น 0 แล้วเรียกให้ SearchRoad หากพบเมืองปลายทาง จะได้ว่า found เป็นจริง ให้แสดงผลโดยเรียกใช้ Display Result (อยู่ในรูปเดียวกันนี้)

ซึ่ง DisplayResult จะนำข้อมูลใน stack ออกแสดงในแบบย้อนกลับ ภาพที่ปรากฏบนจอภาพจะเป็น a b c d เป็นชื่อเมืองที่ผ่านตามที่ต้องการ ขอให้ทดลองรันโปรแกรมนี้ก่อนครับ

ลิสต์ในโปรล็อก

ตามที่กล่าวแล้วว่าลิสต์ในโปรล็อกก็คือสแต็กในภาษาปาสคาล อีกเรื่องหนึ่งที่ผมยังไม่ได้กล่าวถึงคือการดำเนินการกับลิสต์ในโปรล็อกไม่มีอินเด็กซ์สำหรับชี้ ใช้วิธีแยกลิสต์เป็นส่วนหัว (head) กับส่วนหาง (tail) โดยปกติจะเพิ่มหรือถอนข้อมูลกับลิสต์ตรงส่วนหัวเท่านั้น (เช่นเดียวกับสแต็กทำตรง stack pointer เท่านั้น) ซึ่งต้องดำเนินการด้วยการเรียกตนเอง

M

```
BEGIN  { MAIN }
  ClrScr;  RP := 0;  DoAssert;
  REPEAT
    Write(':');
    GetSentence(rdestr,lstch);
    IF  lstch = '?'  THEN  DoQuestion(rdestr)
  UNTIL  rdestr = 'quit.';
END.
```

E2

```
PROCEDURE  SearchRoad(src,dst : NameT);
VAR  dist : Word;  mid : NameT;  idx : Word;
BEGIN
  found := FALSE;
  IF  src = dst  THEN  found := TRUE
  ELSE  IF  FindMid(src,mid,idx)  THEN
    SearchRoad(mid,dst);

  IF  found  THEN
    BEGIN  SP := SP + 1;  stack[SP] := src  END
  ELSE  IF  MoreRoad(src)  THEN  SearchRoad(src,dst);
END;
```

```
PROCEDURE  DisplayResult(sp : Word);
VAR  s,idx,tot : Word;
BEGIN
  FOR  s := sp DOWNTO 1  DO   Write(stack[s],'  ');
  WriteLn;
END;


PROCEDURE  DoQuestion(astr : String);
VAR   srce,dstn : NameT;  dist : Word; idx : Byte;
      aroad : Boolean;
BEGIN
  ClearPass;  idx := 1;  aroad := FALSE;
  srce := GetWord(astr,idx);
  dstn := GetWord(astr,idx);
  IF  srce <> dstn  THEN
    BEGIN
    WHILE  MoreRoad(srce)  DO
      BEGIN  SP := 0;  SearchRoad(srce,dstn);
             IF  found  THEN  DisplayResult(SP)   END;
    END;
  WriteLn('No (more) road');
END;
```

F

ดังตัวอย่างในรูปที่ 8 จะนำข้อมูลในลิสต์เช่น ชื่อเมืองจากต้นลิสต์ (ชั้นบนสุดของสแต็ก) ออกแสดงทีละเมืองจนหมดข้อมูลในลิสต์ ด้วยการถอนส่วนหัวใช้ชื่อตัวแปรว่า H ออกจากลิสต์นำออกแสดงด้วย write ตรงที่เป็นการถอนข้อมูลจริง ๆ คือตรงที่เรียกตนเองด้วยส่วนหางที่เหลือใช้ชื่อตัวแปรว่า T ทำให้ H เดิมหายไป

คือถ้าข้อมูลในลิสต์มีค่าเป็น [a, b, c, d] จะได้ว่า H มีค่าเป็น a และ T มีค่าเป็น b, c, d ในการเรียกตนเองครั้งแรกจะได้ลิสต์ใหม่มีค่าเป็น [b, c, d] และ H กับ T ใหม่มีค่าเป็น b กับ c, d ตามลำดับ

จากความรู้ดังกล่าวมานี้นำไปเขียนโปรแกรมภาษาโปรล็อกได้ดังรูปที่ 9 โดยถ้าถามถึงเส้นทางโดยใช้คำว่า route จะต้องเปลี่ยนคำถามเป็น (ตัวอย่างเช่น)

route (a, y, [a])

ซึ่งไม่สะดวกแก่ผู้ใช้ จึงได้เขียนพรีดิเคต trip เพื่อให้ถามเป็น

trip (a, y)

ขอให้สังเกตว่าได้ให้ชื่อเมืองต้นทางไว้ในลิสต์ตั้งแต่เริ่มต้น ส่งให้แก่ตัวแปร T ในroute เมื่อหาเมืองกลางได้ ได้นำมาเป็นส่วนหัวในการเรียกตนเอง เป็นการเพิ่มข้อมูลให้แก่ลิสต์ ค่าในลิสต์ขณะนี้จะเป็น [b, a]

เมื่อพบเมืองปลายทางคือถึงเมือง y แล้วค่าในลิสต์จะเป็น [y, c, b, a] ให้ writerev แสดงเป็น a b c d ขอให้เทียบ writerew กับ writedir (รูปที่ 8) จะเห็นว่า writedir ให้แสดงผลในช่วงของการเรียกตนเอง ชื่อเมืองจึงเป็นตามลำดับจากหัวไปหางในลิสต์ แต่ writerev ให้แสดงผลในช่วงของการส่งกลับ ชื่อเมืองจึงเรียงลำดับกลับกัน

ประโยชน์ของลิสต์ต่อ

ในส่วนตัวผมเห็นว่าลิสต์เป็นของคู่กับภาษาโปรล็อกไม่น้อยไปกว่าในภาษา LISP แต่เป็นที่น่าแปลกใจว่าหนังสือภาษาอังกฤษเกี่ยวกับเทอร์โบโปรล็อกที่ผมอ้างไว้ท้ายบทความนี้ กลับไม่ค่อยนำลิสต์มาใช้

ได้แสดงให้เห็นประโยชน์ของลิสต์ในการเก็บข้อมูลไว้แสดงผลแล้ว ประโยชน์ของลิสต์มีมากมายอีกเรื่องหนึ่ง คือใช้ในการตรวจสอบเช่น จากโปรแกรมที่ยกมาเป็นตัวอย่างนี้จะเห็นว่า เส้นทางเป็นเส้นทางไปทางเดียว หากต้องการ

```
writedir([]).
writedir([H|T]) :- write(H),
                   writedir(T).
```

รูปที่ 8

```
/* Depth-First Search */
domains
  list = city*
  city = symbol
predicates
  road(city,city)
  route(city,city,list)
  trip(city,city)
  writerev(list)
clauses
  road(a,b).
  road(b,c).
  road(c,d).
  road(a,x).
  road(x,y).
  road(y,z).
  road(c,y).

  route(D,D,L) :- writerev(L),nl.
  route(S,D,T) :-  road(S,M),route(M,D,[M|T]).

  trip(S,D) :- route(S,D,[S]).

  writerev([]).
  writerev([H|T]) :- writerev(T), write(H,' ').
```

รูปที่ 9 โปรแกรมในภาษาโปรล็อก ที่แสดงชื่อเมืองที่จะผ่านด้วย

ให้เป็นไปกลับได้อาจทำได้โดย ให้ข้อมูลไป-กลับ เช่น road (a, b). กับ road (b, a). อีกวิธีหนึ่งคือเขียนโปรแกรมเป็นพรีดิเคต two-way ดังแสดงในรูปที่ 10

ซึ่งไม่ว่าจะเป็นแบบใด จะต้องตรวจสอบว่าได้ผ่านเมืองนี้มาแล้วหรือยัง ไม่เช่นนั้นเส้นทางอาจเป็น a b a b ซ้ำไม่รู้จบได้ ด้วยการใช้พรีดิเคต pass คือเป็นโปรแกรมใหม่ ดังรูปที่ 10

จะตีความหมายของพรีดิเคต route ได้ดังนี้ ถ้าถึงเมืองปลายทางให้แสดงผล ขึ้นบรรทัดใหม่ สำหรับคำว่า fail ขอติดไว้ก่อนครับ ส่วนบรรทัดที่สองจะตีความหมายได้ว่า มีเส้นทางจากเมืองต้นทางไปเมืองปลายทาง ถ้ามีถนนไปกลับ (two-way) จากเมืองต้นทางไปเมืองกลาง และไม่เคยผ่านเมืองกลางมาก่อน (not pass) และมีเส้นทางจากเมืองกลางไปเมืองปลายทาง

พรีดิเคต two-way จะตีความหมายได้ว่า มีถนนไปกลับระหว่างเมือง a กับเมือง b ถ้ามีถนนจากเมือง a ไป b หรือ (ขอให้สังเกตเครื่องหมายด้วย) มีถนนจากเมือง b ไป a

พรีดิเคต pass ซึ่งโดยทั่วไปจะใช้คำว่า member เพราะเป็นการตรวจว่า x เป็นสมาชิกของลิสต์หรือไม่ โดยตรวจว่า x จะเป็นสมาชิก ถ้าเป็นส่วนหัวหรือส่วนหาง

สำหรับคำว่า fail เพื่อให้เทอร์โบโปรล็อกตรวจเส้นทางซ้ำ เช่น เมื่อไม่มีคำว่า fail หากถามว่า trip (z, b) จะได้คำตอบว่า z y x a b หากใช้ fail เทอร์โบโปรล็อกจะตรวจทุกเส้นทางได้คำตอบ z y c b เพิ่มมา

ย้อนกลับไปที่ปาสคาล

ในโพรซิเดอร์ DoQuestion (รูป F) วงรอบ WHILE เป็นตรงที่ผมต้องการทำให้เหมือนกับ fail ในเทอร์โบโปรล็อก คือถ้าคุณตั้งคำถามว่า a y ? จะได้รับคำตอบว่า a b c y กับ a x y เรื่องควรมีเท่านี้ แต่ถ้าถามว่า a z ? จะได้คำตอบว่า a b c z เพียงคำตอบเดียว ทั้งนี้เพราะการหมายรอยเท้าในเส้นทางเส้นนี้ทำให้ตัดเส้นทาง a, x, y, z

นั่นคือในโปรล็อกไม่ได้หมายรอยเท้า ดังที่ผมนำมาใช้ในภาษาปาสคาล โปรล็อกทำอย่างไรจะต้องกล่าวถึงการค้นหาในทางกว้างก่อนครับ รวมทั้งเรื่องที่ผมติดไว้ทั้งหมดก็ต้องขอนำไปกล่าวภายหลังเช่นกัน

นำไปใช้งาน

ผมจะยังไม่กล่าวถึงว่า เราจะนำความรู้ที่ได้จากการศึกษาโปรล็อกด้วยปาสคาลไปใช้ในการเขียนโปรแกรมภาษาโปรล็อกได้อย่างไร แต่จะขอกล่าวถึงการนำโปรแกรมภาษาโปรล็อกที่กล่าวมานี้ไปใช้งานก่อน คือ

- การคมนาคม สามารถสร้างแผนที่เส้นทางคมนาคมได้ทั้งทางบก ทางเรือ และทางอากาศ

- การควบคุมข่ายงาน เช่น PERT และ CPM

- การออกแบบ-ทดสอบวงจรและลายปรินต์

และอื่น ๆ อีกมากคืออะไรก็ได้ที่มีต้นทางปลายทาง แต่ทั้งนี้คงต้องเปลี่ยนชื่อพรีดิเคตใหม่ และปรับปรุงโปรแกรมให้เหมาะสมกับแต่ละงาน

```
/* Depth-First Search 4 */
domains
  list = city*
  city = symbol
predicates
  road(city,city)
  route(city,city,list)
  trip(city,city)
  writerev(list)
  two_way(city,city)
  pass(city,list)
clauses
  road(a,b).
  road(b,c).
  road(c,d).
  road(a,x).
  road(x,y).
  road(y,z).
  road(c,y).

  route(D,D,L) :- writerev(L),nl,fail.
  route(S,D,T) :-  two_way(S,M),
                   not(pass(M,T)),
                   route(M,D,[M|T]).

  trip(S,D) :- route(S,D,[S]).

  writerev([]).
  writerev([H|T]) :- writerev(T), write(H,' ').

  two_way(A,B) :- road(A,B); road(B,A).

  pass(X,[X|_]).
  pass(X,[_|Y]) :- pass(X,Y).
```

รูปที่ 10 โปรแกรมในภาษาโปรล็อก ที่สามารถถามเส้นทางจากปลายทางไปต้นทางได้

หนังสืออ้างอิง

Herbert Schildt, "Advanced Turbo Prolog Version 1.1", Osborne/McGraw-Hill Berkeley, California.

Herbert Schildt, "Advanced Turbo C", Osorne/McGraw-Hill, Berkeley, California.